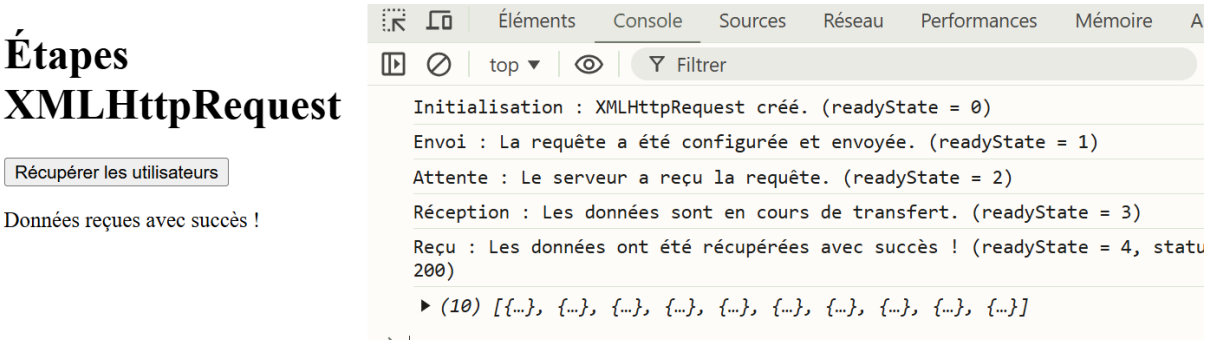
Exercice 1 : Comprendre les étapes d'une requête XMLHttpRequest

Énoncé :

Vous allez explorer les différentes étapes d'une requête `XMLHttpRequest`. Votre tâche est de suivre le processus complet, depuis la création de l'objet jusqu'à la réception des données. Vous afficherez des messages correspondant à chaque étape (`Initialisation`, `Envoi`, `Attente`, `Réception`, `Reçu`).



L'objectif est de :

1. Comprendre les états `readyState` et leurs significations.
2. Gérer l'événement `onreadystatechange` pour détecter et afficher les transitions d'état.
3. Récupérer des données depuis une API publique : **`https://jsonplaceholder.typicode.com/users`**.
4. Afficher les messages correspondants à chaque étape de la requête dans la console et sur la page.

Tâches :

1. Créez une page HTML avec un bouton **"Récupérer les utilisateurs"** et un paragraphe pour afficher les messages.
2. Implémentez une requête `XMLHttpRequest` pour appeler l'API.
3. Capturez et affichez les étapes suivantes :
   - `readyState = 0` : L'objet `XMLHttpRequest` est créé (initialisation).
   - `readyState = 1` : La requête est configurée et prête à être envoyée.
   - `readyState = 2` : Le serveur a reçu la requête.
   - `readyState = 3` : Le serveur transmet les données.
   - `readyState = 4` : La réponse est complètement reçue.
4. Affichez un message d'erreur si la requête échoue (par exemple, statut différent de 200).

Code solution :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Étapes XMLHttpRequest</title>
</head>
<body>
```

```
  <h1>Étapes XMLHttpRequest</h1>
  <button id="fetch-users">Récupérer les utilisateurs</button>
  <p id="message"></p>
  <script>
    // Sélection des éléments
    const button = document.getElementById('fetch-users');
    const message = document.getElementById('message');

    // Ajout d'un événement au bouton
    button.addEventListener('click', () => {
      // Étape 0 : Création de l'objet XMLHttpRequest
      const xhr = new XMLHttpRequest();
      console.log("Initialisation : XMLHttpRequest créé. (readyState = 0)");

      // Gestion des changements d'état
      xhr.onreadystatechange = function () {
        if (xhr.readyState === 1) {
          console.log("Envoi : La requête a été configurée et envoyée. (readyState = 1)");
          message.textContent = "Requête envoyée...";
        }
        if (xhr.readyState === 2) {
          console.log("Attente : Le serveur a reçu la requête. (readyState = 2)");
          message.textContent = "Le serveur traite la demande...";
        }
        if (xhr.readyState === 3) {
          console.log("Réception : Les données sont en cours de transfert. (readyState = 3)");
          message.textContent = "Réception des données...";
        }
        if (xhr.readyState === 4) {
          if (xhr.status === 200) {
            console.log("Reçu : Les données ont été récupérées avec succès ! (readyState = 4, status = 200)");
            message.textContent = "Données reçues avec succès !";
            console.log(JSON.parse(xhr.responseText));
          } else {
            console.error(`Erreur : Statut ${xhr.status}`);
            message.textContent = `Erreur : Impossible de récupérer les données (Statut ${xhr.status})`;
          }
        }
      };

      // Étape 1 : Configuration de la requête
      xhr.open('GET', 'https://jsonplaceholder.typicode.com/users', true);

      // Étape 2 : Envoi de la requête
      xhr.send();
    });
  </script>
</body>
</html>
```

Explications du code :

1. **Création de l'objet `XMLHttpRequest` :**
   - `const xhr = new XMLHttpRequest();`

   - À ce stade, `readyState = 0` et un message est affiché pour indiquer que l'objet est créé.
2. **Gestion de l'événement `onreadystatechange` :**
   - L'événement surveille les changements de l'état `readyState`.
   - Des messages sont affichés pour chaque valeur de `readyState` (de 1 à 4).
3. **Configuration de la requête :**
   - `xhr.open('GET', 'https://jsonplaceholder.typicode.com/users', true);`
   - Cette étape prépare la requête à être envoyée (passage à `readyState = 1`).
4. **Envoi de la requête :**
   - `xhr.send();`
   - L'état passe à `readyState = 2` une fois que le serveur a reçu la requête.
5. **Réception des données :**
   - Lorsque `readyState = 3`, le transfert des données commence.
   - Lors de `readyState = 4`, les données sont entièrement reçues.
6. **Gestion des erreurs :**
   - Vérification du statut HTTP (`xhr.status`).
   - Si le statut est différent de 200, un message d'erreur est affiché.

Exécution :

1. **Lancez le fichier dans votre navigateur.**
2. **Cliquez sur "Récupérer les utilisateurs".**
3. **Observez les messages dans la console et sur la page.**
   - Étape par étape, chaque état sera affiché.
   - Les utilisateurs récupérés s'afficheront dans la console en tant qu'objet JSON.

Résultats attendus :

1. **Console :**
```
Initialisation : XMLHttpRequest créé. (readyState = 0)
Envoi : La requête a été configurée et envoyée. (readyState = 1)
Attente : Le serveur a reçu la requête. (readyState = 2)
Réception : Les données sont en cours de transfert. (readyState = 3)
Reçu : Les données ont été récupérées avec succès ! (readyState = 4, status = 200)
[Affichage des données JSON récupérées]
```

2. **Page Web :**
   - Messages correspondant à chaque étape.
   - Texte final : "Données reçues avec succès !" ou un message d'erreur si la requête échoue.

Exercice 2: Explorer les méthodes et attributs de l'objet XMLHttpRequest

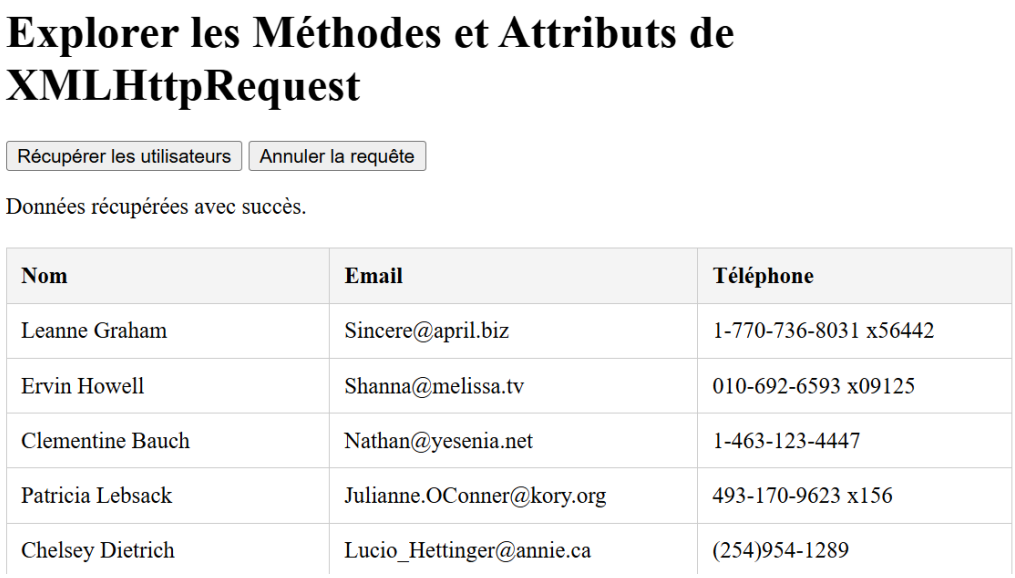
Énoncé :

Dans cet exercice, vous apprendrez à utiliser les principales méthodes et attributs de l'objet `XMLHttpRequest` pour interagir avec une API. Vous devrez créer une page HTML qui :

1. Effectue une requête GET à une API.

2. Affiche les données récupérées dans un tableau.
3. Montre comment utiliser les attributs et méthodes de `XMLHttpRequest` comme :
    - `responseText`
    - `status`
    - `readyState`
    - `setRequestHeader()`
    - `timeout`
    - `abort()`

Vous utiliserez l'API suivante : https://jsonplaceholder.typicode.com/users.



Tâches :

1. **Création d'une requête GET :**
    - Configurez et envoyez une requête GET.
    - Affichez les informations des utilisateurs (nom, email, et téléphone) dans un tableau.
2. **Gestion des attributs et méthodes de XMLHttpRequest :**
    - Utilisez `timeout` pour gérer les délais d'attente.
    - Implémentez un bouton "Annuler la requête" pour tester `abort()`.
3. **Affichez les étapes de la requête en utilisant les états `readyState`.

Code Solution :

```
<!DOCTYPE html>
<!-- Explorer XMLHttpRequest -->
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Explorer XMLHttpRequest</title>
  <style>
    table {
      width: 100%;
      border-collapse: collapse;
      margin: 20px 0;
```

```
    }
    th, td {
      border: 1px solid #ccc;
      padding: 10px;
      text-align: left;
    }
    th {
      background-color: #f4f4f4;
    }
  </style>
</head>
<body>
  <h1>Explorer les Méthodes et Attributs de XMLHttpRequest</h1>
  <button id="fetch-users">Récupérer les utilisateurs</button>
  <button id="abort-request">Annuler la requête</button>
  <p id="status"></p>
  <table id="user-table">
    <thead>
      <tr>
        <th>Nom</th>
        <th>Email</th>
        <th>Téléphone</th>
      </tr>
    </thead>
    <tbody></tbody>
  </table>

  <script>

Votre code ici
  </script>
</body>
</html>
```

Explications :

1. **Création et envoi de la requête :**
   - La méthode `open()` configure la requête en mode GET vers l'API.
   - `send()` envoie la requête.
2. **Gestion des étapes (`readyState`) :**
   - La propriété `readyState` suit les étapes de la requête.
   - Chaque changement d'état est affiché grâce à `onreadystatechange`.
3. **Timeout :**
   - La propriété `timeout` est définie sur 5000 ms.
   - L'événement `ontimeout` gère les cas où la requête dépasse ce délai.
4. **Annulation de la requête :**
   - La méthode `abort()` permet d'annuler une requête en cours.
   - Elle est appelée via le bouton "Annuler la requête".
5. **Affichage des données :**
   - Les données JSON sont converties en tableau HTML dans la fonction `displayUsers`.

Résultats attendus :

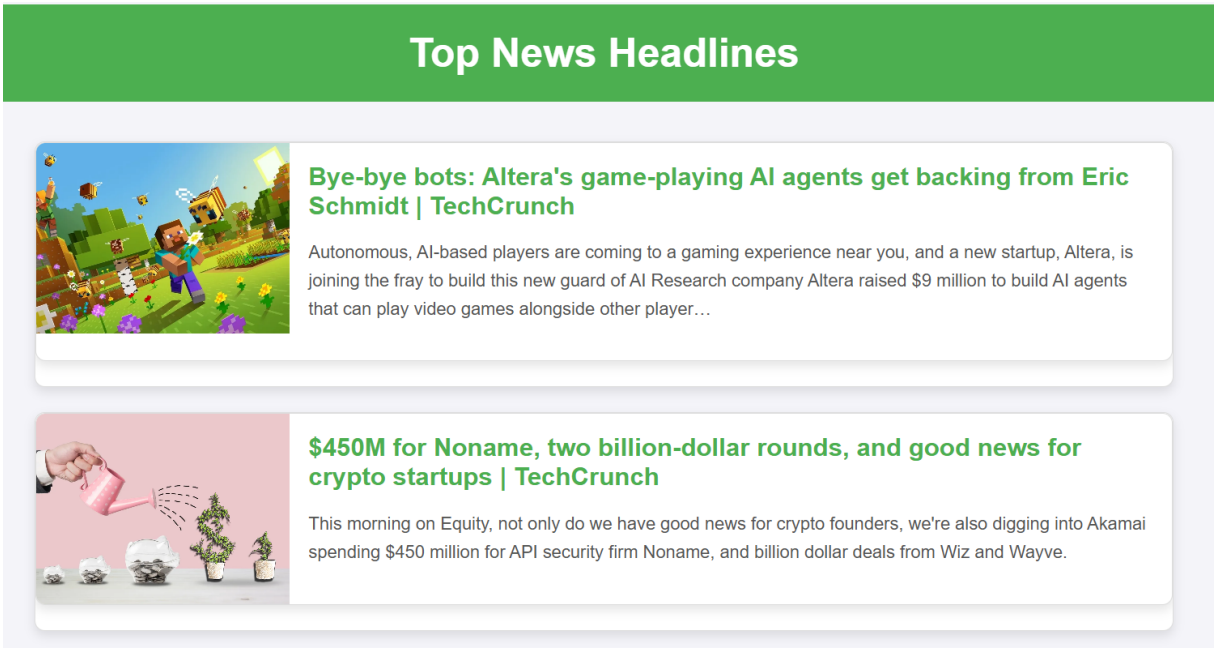
1. **Après un clic sur "Récupérer les utilisateurs" :**
   - Un tableau est rempli avec les noms, emails et téléphones des utilisateurs.
   - Les messages d'état (`readyState`) sont affichés dans l'élément `<p>`.
2. **En cas de délai d'attente dépassé :**
   - Un message indique que le délai est expiré.
3. **Après un clic sur "Annuler la requête" :**
   - La requête est interrompue et un message indique qu'elle a été annulée.

Exercice 3: Affichage des Titres de News avec Images

Objectif :

Dans cet exercice, vous allez créer une page web qui affiche les titres des actualités à partir de l'API NewsAPI. Vous devrez afficher chaque article avec son titre, une image si disponible, et une brève description. Les articles devront être cliquables pour rediriger l'utilisateur vers l'URL de l'article complet.

Your API key is: de674c86f8024e8bae4f3a9f20caaddc



Tâches :

1. Créez une page HTML et récupérez les données des actualités depuis l'API NewsAPI en utilisant `fetch`.
2. Pour chaque article, affichez :
   - Le titre de l'article sous forme de lien cliquable.
   - Une image (si disponible), sinon une image par défaut.
   - Une description brève de l'article.
3. Améliorez l'affichage en utilisant des styles CSS simples pour rendre la page attrayante et lisible.

4. Les articles doivent être affichés sous forme de cartes avec l'image à gauche et le texte à droite.
5. Si une image n'est pas disponible pour un article, un espace réservé (`placeholder`) doit être utilisé à la place.

Prérequis :

- Utilisez l'API de NewsAPI pour récupérer les articles.
- Utilisez `fetch` pour effectuer la requête API et manipuler les données.
- Appliquez des styles CSS pour améliorer l'aspect visuel de la page.

Code complet :

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Top Headlines</title>
 <style>
  body {
   font-family: Arial, sans-serif;
   background-color: #f4f4f9;
   margin: 0;
   padding: 0;
  }
  h1 {
   text-align: center;
   padding: 20px;
   background-color: #4CAF50;
   color: white;
  }
  #news-articles {
   max-width: 900px;
   margin: 20px auto;
   padding: 10px;
  }
  .article {
   background-color: white;
   border: 1px solid #ddd;
   border-radius: 8px;
   margin-bottom: 20px;
   overflow: hidden;
   box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
   display: flex;
   flex-direction: row;
  }
  .article img {
   max-width: 200px;
   object-fit: cover;
   height: 150px;
  }
  .article-content {
   padding: 15px;
   flex: 1;
  }
```

```
  .article h2 {
    font-size: 20px;
    margin: 0 0 10px;
    color: #333;
  }
  .article p {
    color: #555;
    font-size: 14px;
    line-height: 1.6;
  }
  .article a {
    color: #4CAF50;
    text-decoration: none;
  }
  .article a:hover {
    text-decoration: underline;
  }
 </style>
</head>
<body>
 <h1>Top News Headlines</h1>
 <div id="news-articles"></div>

 <script>


Votre code ICI

 </script>
</body>
</html>
```

Explications :

1. **HTML Structure** :
   - La page est structurée avec un titre en haut, suivi d'un conteneur (`#news-articles`) où les articles seront affichés.
   - Chaque article est affiché dans un `div` avec une classe `article` contenant une image et un contenu texte (titre et description).
2. **CSS Styling** :
   - Les articles sont stylisés avec une bordure, un ombrage pour les faire ressortir et un agencement flex pour aligner l'image et le contenu côte à côte.
   - L'image est ajustée à une taille fixe avec `object-fit: cover` pour maintenir son rapport d'aspect.
   - Des couleurs et des effets de survol ont été ajoutés pour rendre les liens plus interactifs.
3. **JavaScript (fetch)** :
   - Le code `fetch` est utilisé pour récupérer les données de l'API NewsAPI.
   - Chaque article est traité pour extraire le titre, l'image (si disponible), et la description. Si aucune image n'est fournie, un `placeholder` est utilisé.

Résultats attendus :

- Une page web qui affiche une liste de titres d'articles de TechCrunch avec leur image et description.
- Si aucune image n'est disponible, une image de remplacement est utilisée.
- Les titres des articles sont des liens cliquables qui ouvrent l'article complet dans un nouvel onglet.

Extension possible :

- Ajouter des filtres de recherche ou de tri pour personnaliser les résultats (par exemple, par catégorie ou par date).
- Ajouter un bouton de pagination pour afficher plus d'articles.

Exercice 4 : Recherche d'Images avec l'API Giphy

Objectif

L'objectif de cet exercice est d'apprendre à utiliser l'**API Giphy** pour rechercher des images à partir d'un mot-clé et afficher une image dans la page web. Vous allez utiliser l'objet `XMLHttpRequest` pour faire une requête HTTP et récupérer une image à partir de l'API Giphy.

API Key = MFah4mEEzRxxtWfCKNNwz3JuWHXyUM6H

Énoncé de l'exercice

Vous devez créer une page HTML permettant à l'utilisateur de saisir un mot-clé dans un champ de texte. Ensuite, en cliquant sur le bouton "Rechercher", l'utilisateur pourra voir une image liée à ce mot-clé provenant de l'API Giphy.

Étapes

1. **Page HTML**
   - Créez un champ de texte où l'utilisateur peut entrer un mot-clé.
   - Ajoutez un bouton qui déclenche la recherche de l'image.
   - Ajoutez une zone où l'image sera affichée après la recherche.

2. **Appel API avec JavaScript**
    - Utilisez l'API Giphy pour rechercher des images à partir du mot-clé saisi par l'utilisateur.
    - Utilisez `XMLHttpRequest` pour envoyer une requête GET à l'API Giphy.
    - L'URL de l'API Giphy est : `https://api.giphy.com/v1/gifs/search?api_key=YOUR_API_KEY&q=KEYWORD&limit=1`
        - Remplacez `YOUR_API_KEY` par votre clé API Giphy personnelle.
        - Remplacez `KEYWORD` par le mot-clé saisi par l'utilisateur.
3. **Afficher l'image**
    - Une fois la réponse reçue de l'API, parsez-la et récupérez l'URL de l'image.
    - Affichez cette image dans la zone dédiée sur la page web.

Code

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <title>Recherche d'Images</title>
</head>
<body>
  <h1>Recherche d'Images avec Giphy</h1>
  <input type="text" id="keyword" placeholder="Entrez un mot-clé">
  <button id="search">Rechercher</button>
  <div id="results"></div>

  <script>
Votre code ici
  </script>
</body>
</html>
```

Explication du code

1. **HTML** :
    - Le champ de texte avec l'ID `keyword` permet à l'utilisateur de saisir un mot-clé pour la recherche.
    - Le bouton avec l'ID `search` déclenche l'événement de recherche.
    - La `div` avec l'ID `results` sert à afficher l'image récupérée.
2. **JavaScript** :
    - Quand l'utilisateur clique sur le bouton "Rechercher", la fonction associée à l'événement `onclick` est exécutée.
    - La valeur du champ de texte est récupérée et utilisée dans l'URL de l'API Giphy pour effectuer la recherche.
    - Une requête `GET` est envoyée à l'API Giphy pour rechercher des images correspondant au mot-clé.
    - Lorsque la réponse est reçue, le code parse la réponse JSON et extrait l'URL de l'image.
    - L'image est ensuite affichée dans la `div results`.

Remarques

- Pour utiliser cette API, vous devez vous inscrire sur le site de Giphy et obtenir une clé API gratuite.
- Cet exercice permet aux étudiants de comprendre comment interagir avec une API externe en utilisant `XMLHttpRequest` pour récupérer et afficher des données dynamiquement dans une page web.

À vous de jouer !

- Remplacez `YOUR_API_KEY` par votre propre clé API et testez la fonctionnalité.

Exercice 5 : Consulter la Météo d'une Ville avec l'API OpenWeatherMap

Objectif

L'objectif de cet exercice est d'apprendre à utiliser l'**API OpenWeatherMap** pour récupérer les données météorologiques d'une ville en fonction du nom saisi par l'utilisateur. Vous allez utiliser l'objet `XMLHttpRequest` pour envoyer une requête à l'API, récupérer les données et afficher la température de la ville.

API Key = 6bfdc0ec4f1017a9ce36bed434e75f82

Énoncé de l'exercice

Vous devez créer une page HTML qui permet à l'utilisateur de saisir le nom d'une ville et d'afficher la température actuelle de cette ville.

1. **Page HTML** :
    - Créez un champ de texte où l'utilisateur pourra entrer le nom d'une ville.
    - Ajoutez un bouton "Obtenir la météo" pour déclencher la récupération des données.
    - Utilisez un script JavaScript pour récupérer la température de la ville via l'API OpenWeatherMap.
2. **Appel à l'API OpenWeatherMap** :

L'URL de l'API est
`https://api.openweathermap.org/data/2.5/weather?q=CITY_NAME&appid=YOUR_API_KEY&units=metric`

- Remplacez `CITY_NAME` par la ville que l'utilisateur entre.
- Remplacez `YOUR_API_KEY` par votre clé API obtenue sur le site d'OpenWeatherMap.

3. **Affichage des données** :
   - Une fois que la réponse est reçue, affichez la température de la ville dans la console du navigateur.

Code

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Météo de votre ville</title>
  <!-- Lien vers Font Awesome pour les icônes -->
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css">
  <!-- Lien vers le fichier CSS externe -->
  <style>
    body {
  font-family: Arial, sans-serif;
  background-color: #f0f4f8;
  margin: 0;
  padding: 0;
}

.container {
  text-align: center;
  padding: 20px;
}

h1 {
  color: #2c3e50;
  font-size: 2rem;
  margin-bottom: 20px;
}

input[type="text"] {
  padding: 10px;
  width: 300px;
  font-size: 1rem;
  margin-right: 10px;
  border: 2px solid #ccc;
  border-radius: 5px;
}

button {
  padding: 10px 20px;
  background-color: #3498db;
  color: white;
  font-size: 1rem;
```

```
    border: none;
    border-radius: 5px;
    cursor: pointer;
}

button:hover {
    background-color: #2980b9;
}

.weather-info {
    display: none;
    margin-top: 20px;
    background-color: #fff;
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.1);
    max-width: 400px;
    margin: 0 auto;
}

#weather h2 {
    color: #3498db;
    font-size: 1.5rem;
}

.details {
    margin-top: 15px;
    font-size: 1.1rem;
}

#weather-icon {
    width: 100px;
    height: 100px;
    margin-top: 20px;
}

    </style>

    <script>
        function getWeather() {
    var city = document.getElementById('city').value;
    var apiKey = '6bfdc0ec4f1017a9ce36bed434e75f82'; // Remplacez par votre propre clé API de OpenWeather
    var url =
` https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${apiKey}&units=metric&lang=fr` ;


votre code ici


    </script>
</head>
<body>
    <div class="container">
        <h1>Vérifiez la météo de votre ville</h1>
        <input type="text" id="city" placeholder="Entrez le nom de la ville...">
        <button onclick="getWeather()">Obtenir la météo</button>
```

```
    <div id="weather-info" class="weather-info">
      <div id="weather">
        <h2></h2>
        <p id="description"></p>
        <div class="details">
          <p><span id="temperature"></span> °C</p>
          <p><span id="humidity"></span>% Humidité</p>
          <p><span id="wind"></span> km/h Vent</p>
        </div>
        <img id="weather-icon" src="" alt="Icone météo">
      </div>
    </div>
  </div>
  <!-- Lien vers le fichier JavaScript -->
</body>
</html>
```

Explication du code

1. **HTML** :
   - Un champ de texte avec l'ID `city` permet à l'utilisateur de saisir le nom d'une ville.
   - Un bouton avec l'ID `getWeather` permet de déclencher la recherche des données météo.
2. **JavaScript** :
   - Lorsque l'utilisateur clique sur le bouton "Obtenir la météo", la fonction associée à l'événement `onclick` est exécutée.
   - La valeur de la ville saisie est récupérée et utilisée dans l'URL de l'API pour effectuer la recherche des données météo.
   - Une requête HTTP de type `GET` est envoyée à l'API OpenWeatherMap pour récupérer les données météorologiques de la ville spécifiée.
   - Lorsque la réponse est reçue (et si le statut de la requête est `200`), les données sont extraites et la température de la ville est affichée dans la console.

Remarques

- Vous devez vous inscrire sur OpenWeatherMap pour obtenir une clé API gratuite.
- Le script récupère la température en Celsius (`units=metric`), mais vous pouvez aussi choisir d'autres unités comme Fahrenheit ou Kelvin en modifiant cette partie de l'URL.

À vous de jouer !

- Remplacez `YOUR_API_KEY` par votre propre clé API et testez le code pour obtenir la température de différentes villes.

Exercice 6. USGS Earthquake API

Description : Cette API fournit des informations sur les séismes en temps réel, y compris la magnitude, la localisation et le temps des derniers séismes.

Derniers Séismes

- **Lieu:** 58 km ESE of Pedro Bay, Alaska
  Magnitude: 1.9
  Heure: 12/3/2024, 9:43:01 PM
- **Lieu:** 8 km S of Fern Forest, Hawaii
  Magnitude: 1.78
  Heure: 12/3/2024, 9:32:32 PM
- **Lieu:** 2 km ENE of Winchester, CA
  Magnitude: 1.31
  Heure: 12/3/2024, 9:28:36 PM
- **Lieu:** Pagan region, Northern Mariana Islands
  Magnitude: 4.9
  Heure: 12/3/2024, 9:27:38 PM
- **Lieu:** 55 km NNW of Petersville, Alaska
  Magnitude: 1.8
  Heure: 12/3/2024, 9:22:47 PM

Exemple en HTML et JavaScript :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>USGS Earthquake Data</title>
</head>
<body>
    <h1>Derniers Séismes</h1>
    <div id="earthquake-list"></div>

    <script>

   Votre code ici
    </script>
</body>
</html>
```

Exercice 7. Global Disaster Alert and Coordination System (GDACS)

Description : Cette API fournit des alertes en temps réel pour les catastrophes naturelles comme les séismes, les tsunamis, et les cyclones.

Utiliser cette URL pour récupérer les données:

https://www.gdacs.org/gdacsapi/api/events/geteventlist/ARCHIVE?eventlist=EQ;TC;FL;VO;WF

Utiliser ce site pour formater la repense JSON : *https://codebeautify.org/jsonviewer*

Événements GDACS

Type	Nom	Pays	Date de début	Date de fin	Description	Alert Level
TC	FENGAL-24	India	29/11/2024 18:00:00	01/12/2024 06:00:00	Tropical Cyclone FENGAL-24	Orange
TC	MAN-YI-24	Philippines, China, Guam	09/11/2024 00:00:00	19/11/2024 12:00:00	Tropical Cyclone MAN-YI-24	Red
TC	USAGI-24	Philippines, Taiwan	11/11/2024 00:00:00	16/11/2024 06:00:00	Tropical Cyclone USAGI-24	Red
TC	TORAJI-24	Philippines, China	09/11/2024 06:00:00	14/11/2024 00:00:00	Tropical Cyclone TORAJI-24	Orange
VO	Lewotobi	Indonesia	14/10/2024 18:40:00	13/11/2024 00:00:00	Eruption Lewotobi	Orange
TC	YINXING-24	Philippines, Viet Nam	03/11/2024 06:00:00	12/11/2024 12:00:00	Tropical Cyclone YINXING-24	Red
FL	Non disponible	Philippines	07/11/2024 01:00:00	12/11/2024 01:00:00	Flood in Philippines	Orange
TC	RAFAEL-24	Cuba, Jamaica	03/11/2024 21:00:00	10/11/2024 21:00:00	Tropical Cyclone RAFAEL-24	Orange
FL	Non disponible	Spain	27/10/2024 15:00:00	04/11/2024 11:00:00	Flood in Spain	Orange
TC	KONG-REY-24	Philippines, Taiwan, China, Japan	25/10/2024 00:00:00	01/11/2024 18:00:00	Tropical Cyclone KONG-REY-24	Red

Code :

```
<!DOCTYPE html>
<html lang="fr">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Événements GDACS</title>
</head>

<body>
  <h1>Événements GDACS</h1>
  <table border="1">
    <thead>
      <tr>
        <th>Type</th>
        <th>Nom</th>
        <th>Pays</th>
        <th>Date de début</th>
        <th>Date de fin</th>
        <th>Description</th>
        <th>Alert Level</th>
      </tr>
    </thead>
    <tbody id="gdacs-data">
      <tr>
        <td colspan="7">Chargement des données...</td>
      </tr>
```

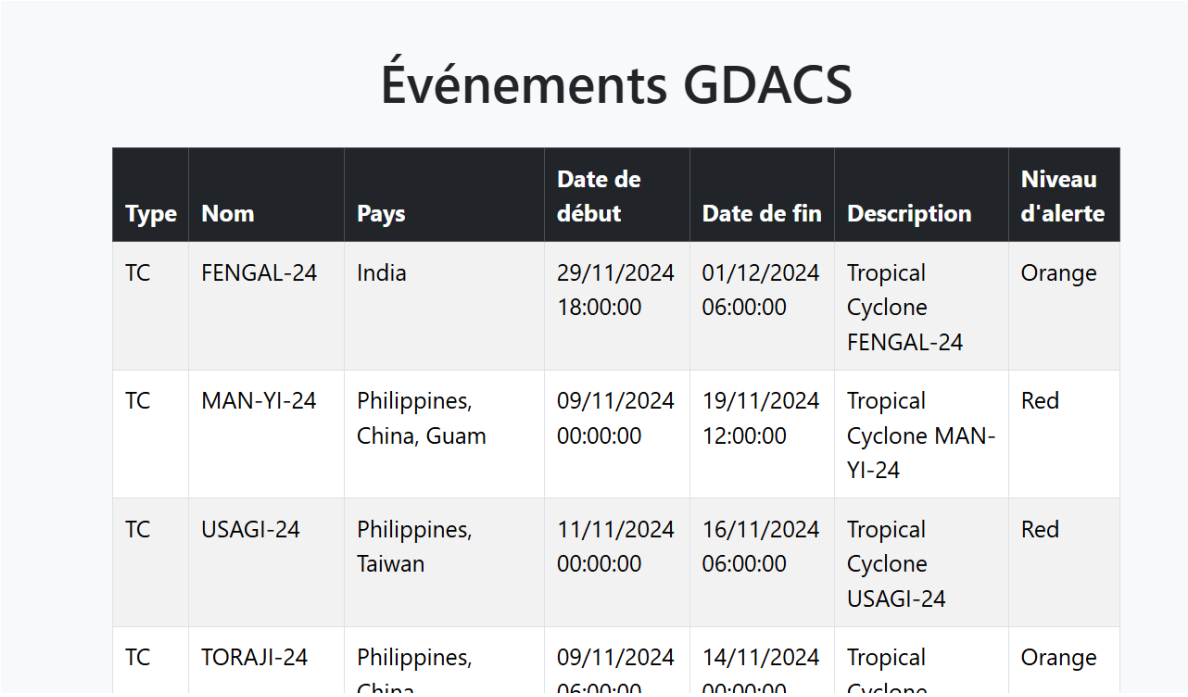
```
    </tbody>
  </table>

  <script>
    const url =
"https://www.gdacs.org/gdacsapi/api/events/geteventlist/ARCHIVE?eventlist=EQ;TC;FL;VO;WF";

    fetch(url)
      .then(response => response.json())
      .then(data => {
        const tbody = document.getElementById('gdacs-data');
        tbody.innerHTML = ""; // Vider le contenu par défaut

        // Vérifier si des événements existent
        if (data.features && data.features.length > 0) {
          data.features.forEach(event => {
            const props = event.properties || {}; // Propriétés de l'événement
            const type = props.eventtype || "Non disponible";
            const name = props.eventname || "Non disponible";
            const country = props.country || "Non précisé";
            const fromDate = props.fromdate ? new Date(props.fromdate).toLocaleString() : "Non précisée";
            const toDate = props.todate ? new Date(props.todate).toLocaleString() : "Non précisée";
            const description = props.description || "Non disponible";
            const alertLevel = props.alertlevel || "Non précisé";

            tbody.innerHTML += `
              <tr>
            Votre code ici

              </tr>
            `;
          });
        } else {
          tbody.innerHTML = `
            <tr>
              <td colspan="7">Aucun événement trouvé</td>
            </tr>
          `;
        }
      })
      .catch(error => {
        console.error("Erreur lors de la récupération des données : ", error);
        document.getElementById('gdacs-data').innerHTML = `
          <tr>
            <td colspan="7">Erreur lors de la récupération des données</td>
          </tr>
        `;
      });
  </script>
</body>

</html>
```

Exercice 8: Amélioration de l'exercice précèdent

```
<!DOCTYPE html>
<html lang="fr">
```

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Événements GDACS</title>
  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">
</head>

<body class="bg-light">
  <div class="container my-5">
    <h1 class="text-center mb-4">Événements GDACS</h1>

    <table class="table table-striped table-bordered">
      <thead class="table-dark">
        <tr>
          <th>Type</th>
          <th>Nom</th>
          <th>Pays</th>
          <th>Date de début</th>
          <th>Date de fin</th>
          <th>Description</th>
          <th>Niveau d'alerte</th>
        </tr>
      </thead>
      <tbody id="gdacs-data">
        <tr>
          <td colspan="7" class="text-center">Chargement des données...</td>
        </tr>
      </tbody>
    </table>
  </div>

  <!-- JavaScript -->
  <script>
    const url =
"https://www.gdacs.org/gdacsapi/api/events/geteventlist/ARCHIVE?eventlist=EQ;TC;FL;VO;WF";

    // Fonction pour obtenir une icône Bootstrap basée sur le niveau d'alerte
    function getAlertIcon(alertLevel) {
      switch (alertLevel.toLowerCase()) {
        case 'green':
          return '<i class="bi bi-circle-fill" style="color: green;"></i>'; // Vert
        case 'orange':
          return '<i class="bi bi-circle-fill" style="color: orange;"></i>'; // Orange
        case 'red':
          return '<i class="bi bi-circle-fill" style="color: red;"></i>'; // Rouge
        default:
          return '<i class="bi bi-dash-circle" style="color: gray;"></i>'; // Niveau non défini
      }
    }

    fetch(url)
      .then(response => response.json())
      .then(data => {
        const tbody = document.getElementById('gdacs-data');
        tbody.innerHTML = ""; // Vider le contenu par défaut
```

```
        if (data.features && data.features.length > 0) {
          data.features.forEach(event => {
            const props = event.properties || {};
            const type = props.eventtype || "Non disponible";
            const name = props.eventname || "Non disponible";
            const country = props.country || "Non précisé";
            const fromDate = props.fromdate ? new Date(props.fromdate).toLocaleString() : "Non précisée";
            const toDate = props.todate ? new Date(props.todate).toLocaleString() : "Non précisée";
            const description = props.description || "Non disponible";
            const alertLevel = props.alertlevel || "Non précisé";

            tbody.innerHTML += `
              <tr>
                <td>${type}</td>
                <td>${name}</td>
                <td>${country}</td>
                <td>${fromDate}</td>
                <td>${toDate}</td>
                <td>${description}</td>
                <td>${getAlertIcon(alertLevel)} ${alertLevel}</td>
              </tr>
            `;
          });
        } else {
          tbody.innerHTML = `
            <tr>
              <td colspan="7" class="text-center">Aucun événement trouvé</td>
            </tr>
          `;
        }
      })
      .catch(error => {
        console.error("Erreur lors de la récupération des données : ", error);
        document.getElementById('gdacs-data').innerHTML = `
          <tr>
            <td colspan="7" class="text-center text-danger">Erreur lors de la récupération des données</td>
          </tr>
        `;
      });
  </script>

  <!-- Bootstrap JS & Icons -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/bootstrap-icons/font/bootstrap-icons.js"></script>
</body>

</html>
```



Événements GDACS

Type	Nom	Pays	Date de début	Date de fin	Description	Niveau d'alerte
TC	FENGAL-24	India	29/11/2024 18:00:00	01/12/2024 06:00:00	Tropical Cyclone FENGAL-24	Orange
TC	MAN-YI-24	Philippines, China, Guam	09/11/2024 00:00:00	19/11/2024 12:00:00	Tropical Cyclone MAN-YI-24	Red
TC	USAGI-24	Philippines, Taiwan	11/11/2024 00:00:00	16/11/2024 06:00:00	Tropical Cyclone USAGI-24	Red
TC	TORAJI-24	Philippines,	09/11/2024	14/11/2024	Tropical	Orange

Exercice 9. OpenStreetMap Nominatim API

Description : Cette API permet de rechercher des lieux et d'obtenir leurs coordonnées géographiques (latitude, longitude).

Géolocalisation d'un Lieu

Béja Trouver

Lieu: Béja, باجة, معتمدية باجة الشمالية, Béja, 9000, Tunisia
Latitude: 36.7236755
Longitude: 9.185382

code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Recherche de Lieu - Nominatim</title>
</head>
<body>
  <h1>Géolocalisation d'un Lieu</h1>
  <input type="text" id="place" placeholder="Entrez un lieu" />
  <button onclick="getLocation()">Trouver</button>
  <div id="location"></div>

  <script>
Votre code ici
  </script>
```

```
</body>
</html>
```

---

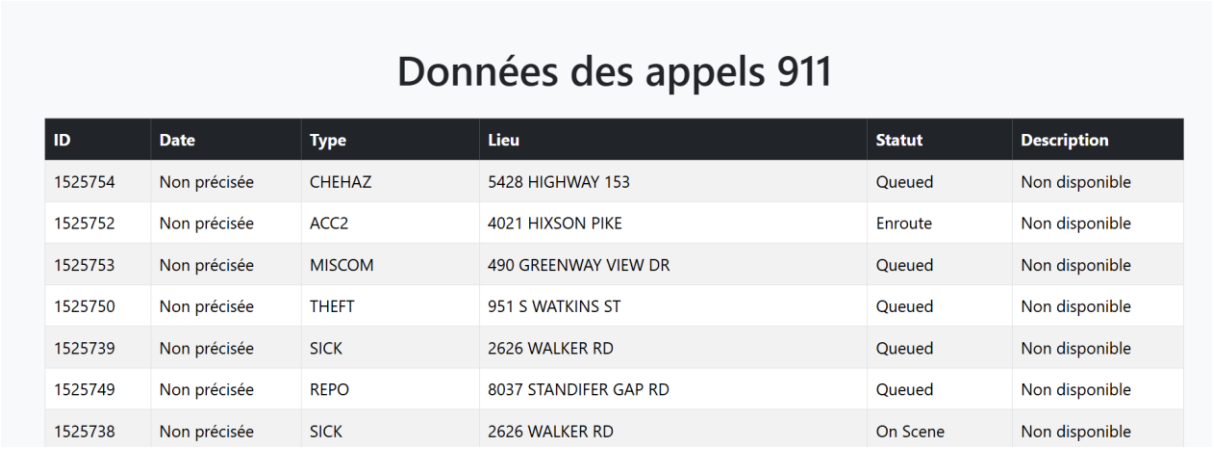
10. 911 API

Description : Cette API fournit des informations en temps réel sur les appels d'urgence et les interventions, principalement aux États-Unis.

https://hc911server.com/api/calls

Champs affichés :

- **ID** : Identifiant de l'appel.
- **Date** : Date et heure de l'appel.
- **Type** : Type d'incident (par exemple, incendie, médical, etc.).
- **Lieu** : Emplacement de l'incident.
- **Statut** : Statut actuel (en cours, résolu, etc.).
- **Description** : Détails supplémentaires de l'incident.



Données des appels 911

ID	Date	Type	Lieu	Statut	Description
1525754	Non précisée	CHEHAZ	5428 HIGHWAY 153	Queued	Non disponible
1525752	Non précisée	ACC2	4021 HIXSON PIKE	Enroute	Non disponible
1525753	Non précisée	MISCOM	490 GREENWAY VIEW DR	Queued	Non disponible
1525750	Non précisée	THEFT	951 S WATKINS ST	Queued	Non disponible
1525739	Non précisée	SICK	2626 WALKER RD	Queued	Non disponible
1525749	Non précisée	REPO	8037 STANDIFER GAP RD	Queued	Non disponible
1525738	Non précisée	SICK	2626 WALKER RD	On Scene	Non disponible

Code :

```
<!DOCTYPE html>
<html lang="fr">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Appels 911</title>
  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">
</head>

<body class="bg-light">
  <div class="container my-5">
    <h1 class="text-center mb-4">Données des appels 911</h1>

    <table class="table table-striped table-bordered">
      <thead class="table-dark">
        <tr>
```

```
            <th>ID</th>
            <th>Date</th>
            <th>Type</th>
            <th>Lieu</th>
            <th>Statut</th>
            <th>Description</th>
          </tr>
        </thead>
        <tbody id="calls-data">
          <tr>
            <td colspan="6" class="text-center">Chargement des données...</td>
          </tr>
        </tbody>
      </table>
  </div>

  <!-- JavaScript -->
  <script>
    const apiUrl = "https://hc911server.com/api/calls";

votre code ici
  </script>

  <!-- Bootstrap JS -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"></script>
</body>

</html>
```

11. Health API (Open Health Data)

Description : Cette API permet d'obtenir des informations sur les hôpitaux, les établissements de santé et les services d'urgence.

Utiliser ce lien : https://healthdata.gov/dataset/Healthcare-Finder-API-7wj9-8ch6-Archive-Repository/ikvk-a35e/data_preview

Lien API : https://healthdata.gov/resource/ikvk-a35e.json

Code :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Health Data Table</title>
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css">
</head>
<body>
<div class="container mt-5">
  <h2 class="mb-4">Health Data Table</h2>
  <table class="table table-bordered table-hover">
    <thead class="table-dark">
```

```
        <tr>
          <th>#</th>
          <th>Date de mise à jour</th>
          <th>Jours depuis mise à jour</th>
          <th>Utilisateur</th>
          <th>Archive</th>
        </tr>
      </thead>
      <tbody id="health-data">
        <!-- Contenu généré dynamiquement -->
      </tbody>
    </table>
</div>

<script>
    const data = [
      {
        "update_date": "2021-02-25T20:57:12.000",
        "days_since_update": "0",
        "user": "Tyler Service Account - HHS",
        "archive_link": {
          "url": "https://finder.healthcare.gov/%23services"
        }
      },
      {
        "update_date": "2021-03-12T18:09:17.000",
        "days_since_update": "15",
        "user": "Tyler Service Account - HHS",
        "archive_link": {
          "url": "https://finder.healthcare.gov/%23services"
        }
      }
    ];

    const tbody = document.getElementById('health-data');
    data.forEach((item, index) => {
      const updateDate = item.update_date || "Non disponible";
      const daysSinceUpdate = item.days_since_update || "Non disponible";
      const user = item.user || "Non spécifié";
      const archiveUrl = item.archive_link?.url || "#";

      tbody.innerHTML += `
        <tr>
          <td>${index + 1}</td>
          <td>${updateDate}</td>
          <td>${daysSinceUpdate}</td>
          <td>${user}</td>
          <td><a href="${archiveUrl}" class="btn btn-primary btn-sm" target="_blank">Voir</a></td>
        </tr>
      `;
    });
</script>
</body>
</html>
```